JTRS SINCGARS Physical API Service Definition

V1.0
December 15, 2000

Prepared for the
Joint Tactical Radio System (JTRS) Joint Program Office

Prepared by the
Modular Software-programmable Radio Consortium
Under Contract No. DAAB15-00-3-0001

Revision Summary

1.0	Initial release

Table Of Contents

List of Figures

List of Tables

1 INTRODUCTION.

1.1 OVERVIEW.

The SINCGARS Physical application-program interface (API) provides a standardized interface to the Physical Layer. The Physical Layer Services are grouped into real time and non-real time Service Groups. The non-real-time Physical-Layer services provide Service Users with methods to send non-real-time configuration and control data into the Physical Layer. Real-time control and real-time signals are time critical signals to and from the Physical Layer.

Location of the Physical Layer, with respect to other JTRS layers, is shown in Figure 1-1. The Physical non-real-time is the "**B**" interface into the Physical Layer. The Physical real-time is the "**A**" interface. In reality, these are the same interface, but it helps conceptually to think of them as different interfaces based on the type of service provided.

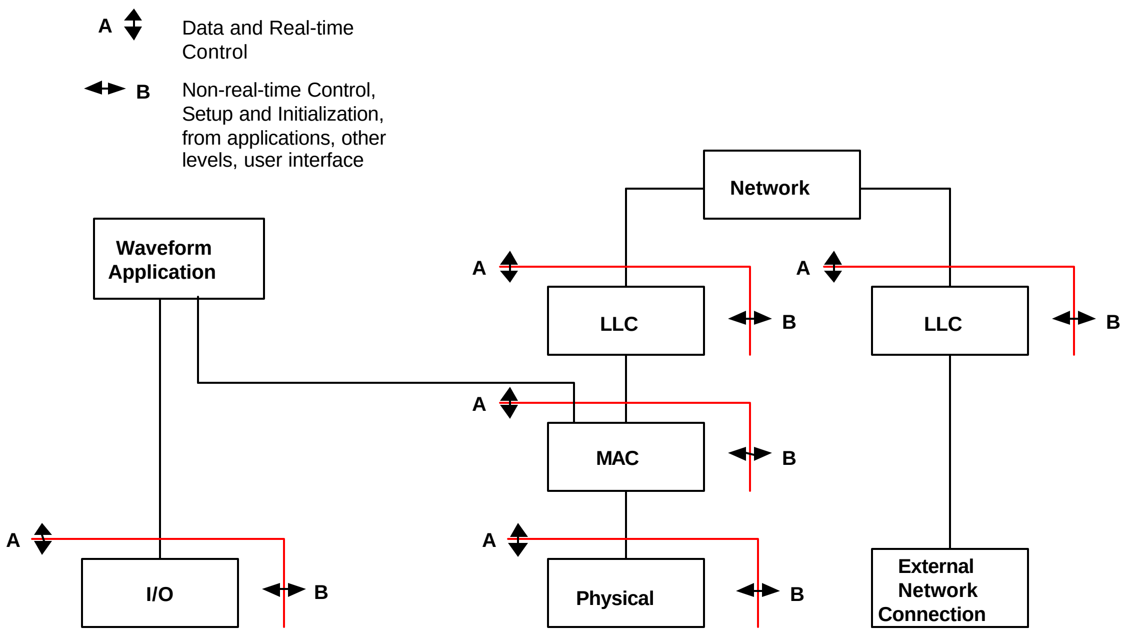
A
Data and Real-time Control
B
Non-real-time Control, Setup and Initialization, from applications, other levels, user interface
Network
Waveform Application
A
A
LLC
B
LLC
B
A
MAC
B
A
A
I/O
B
Physical
B
External Network Connection

Figure 1-1. Service Definition Overview

Reader Notes:

1. In the current version of this document, exceptions are not fully specified. This should not be interpreted to mean that methods cannot raise exceptions. Exceptions will be defined as the design progresses to conform with SCA requirements and good SW Engineering practice and be documented in a later version of this document.
2. This document contains TBDs where values cannot be determined this early in the design process.

1.2 SERVICE LAYER DESCRIPTION.

The primary Physical Layer Service User is the MAC Layer.

This part of the SINCGARS API is realized by instantiating the SCA Physical non-real-time, Physical real-time, and Packet Building Blocks using SINCGARS specific parameters. In general, the Physical Layer is responsible for the modulation and demodulation of data and for transmitting bits over-the-air. The following Service Groups are included in the Physical Layer.

- Physical non-real time:
    - antenna control
    - transceiver setup
    - modulation setup
    - radio mode
    - receive termination
    - transmit inhibit
    - physical management.
- Physical real time:
    - Transmit Packet (data movement into Physical Layer toward antenna)
    - Receive Packet (data movement out of the Physical Layer away from antenna)
    - Flow Control.

Each Service Group will be described in detail.

1.3 MODES OF SERVICE.

This API is used for all transmit and receive modes and for some BIT modes.

1.4 SERVICE STATES.

The API is used for all modes of operations except standby.

1.5 REFERENCED DOCUMENTS.

Document No.	Document Title
MSRC-5000SCA	Software Communications Architecture Specification
MSRC-5000API	Application Program Interface Supplement to the Software Communications Architecture Specification, Appendix C Generic Packet Building Block Service Definition
MSRC-5000API	Application Program Interface Supplement to the Software Communications Architecture Specification, Appendix E Physical Non-Real-Time Building Block Service Definition
MSRC-5000API	Application Program Interface Supplement to the Software Communications Architecture Specification, Appendix D Physical Real-Time Building Block Service Definition

2 UUID.

The UUID for this API is `62fb9f70-d1d3-11d4-8cc8-00104b23b8a2`.

3 SERVICES.

The SINCGARS Physical interface is defined in terms of the services provided by the Service Provider, and the individual primitives that may flow between the Service User and Service Provider.

The services are tabulated in Table 1 and described more fully in the remainder of this section and section 4.

Table 1. Cross-Reference of Services and Primitives

NON-REAL TIME SERVICES		
Service Group	**Service**	**Primitives or Structure Attributes**
Antenna Control	Set Receive Antenna	setRxAntenna(Antenna : in SINCGARSAntennaType) : boolean
	Set Transmit Antenna	setTxAntenna(Antenna: in SINCGARSAntennaType) : boolean
Transceiver Setup	Set Up Receiver Parameters	setUpReceiverParams(RecvParams : in SINCGARSRecvParamsType) : boolean
	Set Up Transmetter Parameters	setUpTransmitterParams(TransParams : in SINCGARSTransParamsType) : boolean
	Get BIT Results	getBIT(BITResult : out SINCGARSAPI::CommonTypes::OctetSequence) : Void
	Perform BIT test	performBIT(BITIdentifier : in unsigned long) : void
Modulation Setup	Set Up Receiver Modulation	setUpReceiverModulation(RecvMod : in SINCGARSRecvModType) : boolean
	Set Up Transmitter Modulation	setUpTransmitterModulation(TransMod : SINCGARSTransModType) : boolean
Radio Mode	Set Radio Mode	setRadioMode(RadioMode : in SINCGARSRadioModeType) : boolean
Receive Termination	Drop Capture	dropCapture( ) : boolean
	Abort Receive	abortReceive( ) : boolean
Transmit Inhibit	Inhibit Transmit	inhibitTransmit(Inhibit : in boolean) : boolean
Physical Management	Maximum Transmission Unit	getMaxTU( ) : unsigned short
	Minimum Transmission Unit	getMinTU( ) : unsigned short

Table 1. Cross-Reference of Services and Primitives - Continued

REAL TIME SERVICES		
Service Group	**Service**	**Primitives or Structure Attributes**
Transmit Packet (packets sent from MAC Layer to Physical Layer)	DownStreamPacket	*pushPacket*(priority : in octet, transmitControl : in PhysicalDownStreamControlType, payload : in SINCGARSAPI::CommonTypes::OctetSequence)void
	PhysicalDownstreamControlType	hopTimeOfFirstSymbol, frequencyInHz, StreamControlType
	StreamControlType	EndOfStream, StreamID, SequenceNum,
Receive Packet (received packets sent from Physical Layer to MAC Layer.	UpStreamPacket	*pushPacket*(priority : octet, control : in PhysicalUpstreamControlType, payload : in SINCGARSAPI::CommonTypes::OctetSequence)
	PhysicalUpstreamControlType	hopTimeOfFirstSymbol, noiseSignalStrength, receiveSignalStrength, transmitSignalStrength, streamControlType
	StreamControlType	EndOfStream, StreamID, SequenceNum,
Flow Control (upstream & downstream)	PacketQueueSetUp	enableFlowControlSignals(enable : in boolean) : void enableEmptySignal(enable : boolean) : void spaceAvailable(priorityQueueID : in octet) : unsigned short setNumOfPriorityQueues(numOfPriorities : in octet) : void
	PacketQueueSignals	signalHighWatermark(priorityQueueID : in octet) void signalLowWaterMark(priorityQueueID : in octet) void signalEmpty( )void

3.1 NON-REAL-TIME SERVICES.

3.1.1 Antenna Control.

Antenna control selects which antennas are to be connected to the SINCGARS waveform. Refer to Figure 3-1 and the following paragraphs to describe the antenna control process.

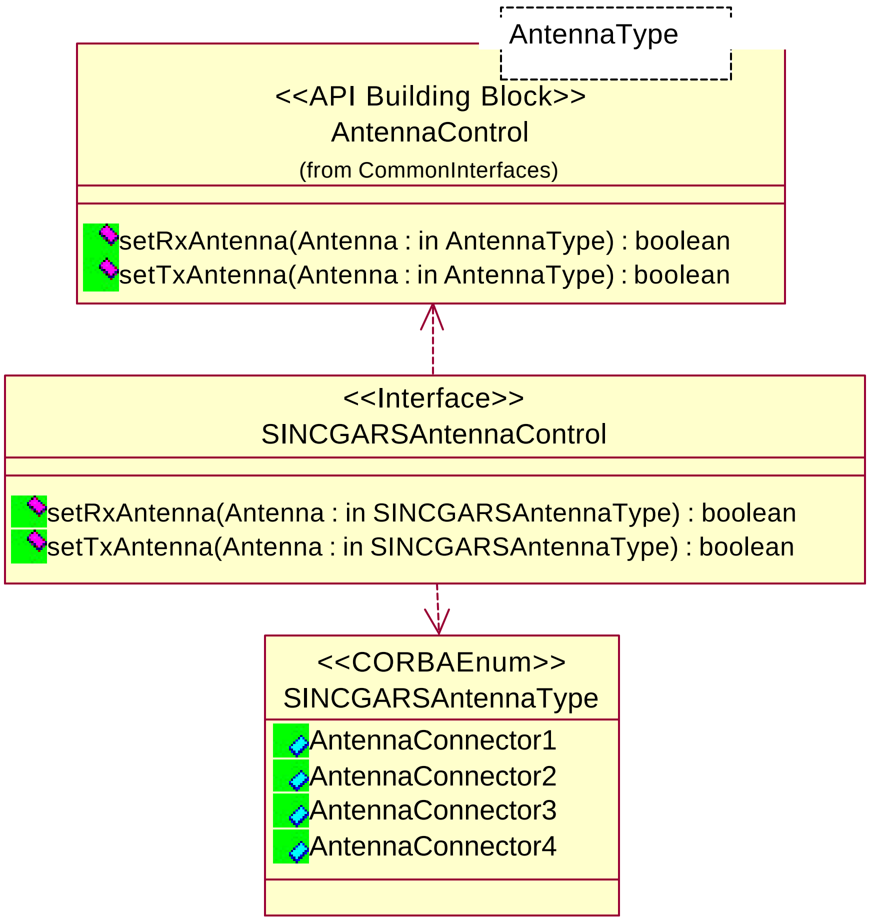
AntennaType
<<API Building Block>>
AntennaControl
(from CommonInterfaces)
setRxAntenna(Antenna : in AntennaType) : boolean
setTxAntenna(Antenna : in AntennaType) : boolean
<<Interface>>
SINCGARSAntennaControl
setRxAntenna(Antenna : in SINCGARSAntennaType) : boolean
setTxAntenna(Antenna : in SINCGARSAntennaType) : boolean
<<CORBAEnum>>
SINCGARSAntennaType
AntennaConnector1
AntennaConnector2
AntennaConnector3
AntennaConnector4

Figure 3-1. SINCGARS Antenna Control

3.1.1.1 setRxAntenna Service.

The setRxAntenna Service allows the radio to select the receive antenna. The receive antenna may be the same as the transmit antenna or may be different.

3.1.1.2 setTxAntenna Service.

The setTxAntenna Service allows the radio to select the transmit antenna. The transmit antenna may be the same as the receive antenna or may be different.

3.1.2 TransceiverSetup.

This class sets up parameters in the Physical Layer which are not modulation or real-time dependent. Refer to Figure 3-2.

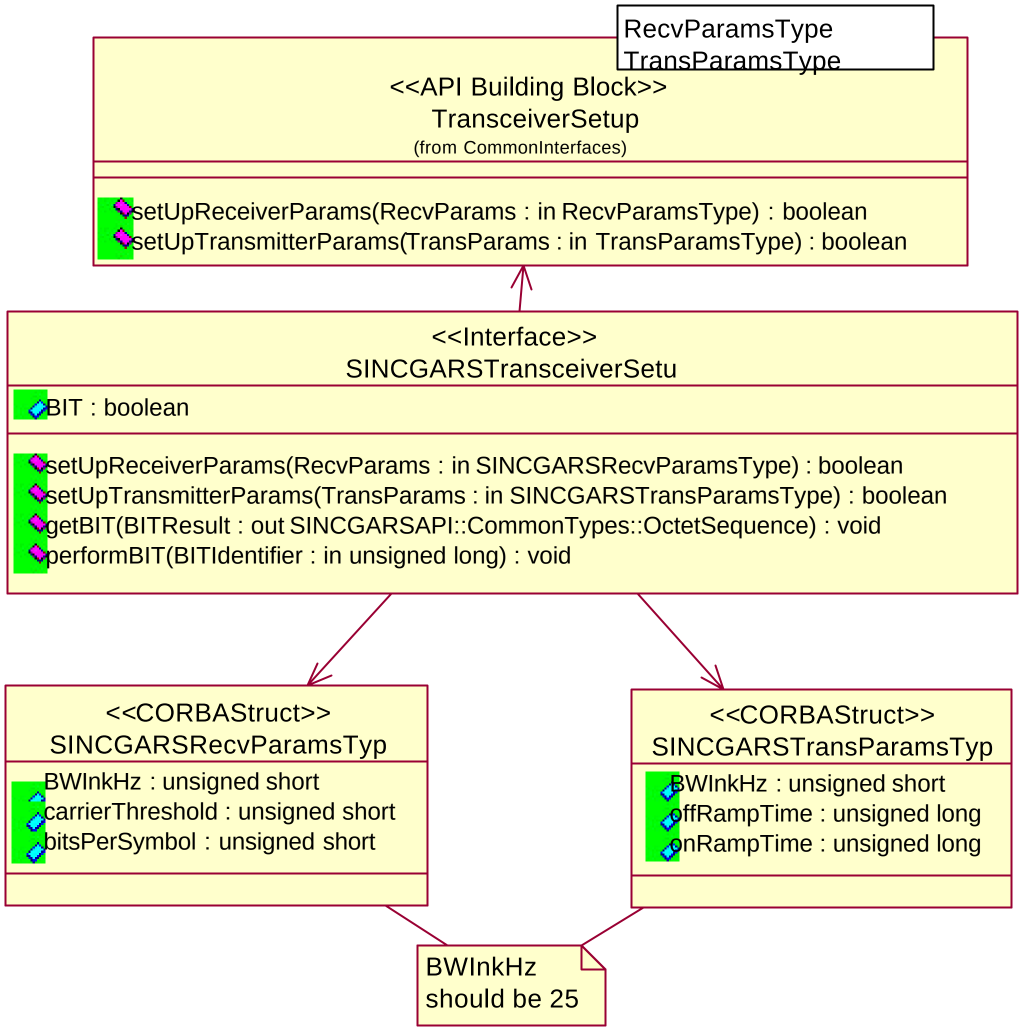
RecvParamsType
TransParamsType
<<API Building Block>>
TransceiverSetup
(from CommonInterfaces)
setUpReceiverParams(RecvParams : in RecvParamsType) : boolean
setUpTransmitterParams(TransParams : in TransParamsType) : boolean
<<Interface>>
SINCGARSTransceiverSetu
BIT : boolean
setUpReceiverParams(RecvParams : in SINCGARSRecvParamsType) : boolean
setUpTransmitterParams(TransParams : in SINCGARSTransParamsType) : boolean
getBIT(BITResult : out SINCGARSAPI::CommonTypes::OctetSequence) : void
performBIT(BITIdentifier : in unsigned long) : void
<<CORBAStruct>>
SINCGARSRecvParamsTyp
BWInkHz : unsigned short
carrierThreshold : unsigned short
bitsPerSymbol : unsigned short
<<CORBAStruct>>
SINCGARSTransParamsTyp
BWInkHz : unsigned short
offRampTime : unsigned long
onRampTime : unsigned long
BWInkHz
should be 25

Figure 3-2. SINCGARS Transceiver Setup

3.1.2.1 setUpReceiverParams.

setUpReceiverParams sets the receive channel bandwidth to 25 kHz, sets carrier threshold to **TBD**, and sets bits per symbol to **TBD**.

3.1.2.2 setUpTransmitterParams.

sSetUpTransmitterParams sets the transmit channel bandwidth to 25 kHz, sets offRampTime to **TBD** microseconds, and sets onRampTime to **TBD** microseconds.

3.1.2.3 getBIT Results.

getBIT returns the results of the last execution of performBIT .

3.1.2.4 performBIT.

peformBIT invokes BIT at the Physical Layer.

3.1.3 ModulationSetup.

ModulationSetup (Figure 3-3) selects the SINCGARS modulation and demodulation methods and sets the required parameters for both.

3.1.3.1 setUpReceiverModulation.

setUpReceiverModulation sets the demodulator to receive FSK alone, FM alone, or either FSK or FM whichever is received. FM modulation is for single channel, plain text voice only. This method provides the peak-to-peak deviation for FSK and FM, and the subaudible tone deviation for SC PT Voice.

3.1.3.2 setUpTransmitterModulation.

setUpTransmitterModulation sets the modulator for FSK or FM. FM modulation is for single channel, plain text voice only. This method provides the peak-to-peak deviation for FSK and FM, and the subaudible tone deviation for SC PT Voice.

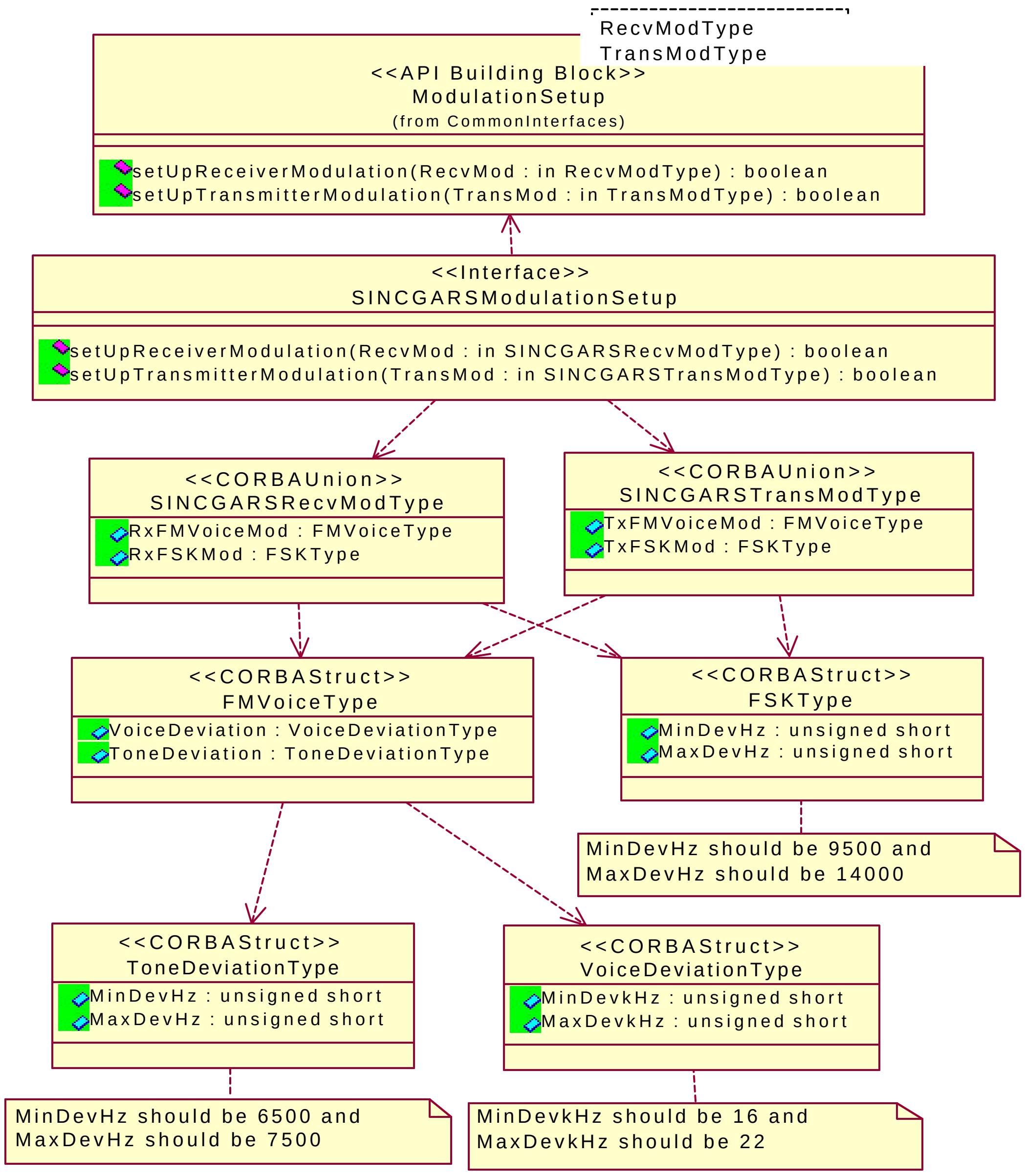
RecvModType
TransModType
<<API Building Block>>
ModulationSetup
(from CommonInterfaces)
setUpReceiverModulation(RecvMod : in RecvModType) : boolean
setUpTransmitterModulation(TransMod : in TransModType) : boolean
<<Interface>>
SINCGARSModulationSetup
setUpReceiverModulation(RecvMod : in SINCGARSRecvModType) : boolean
setUpTransmitterModulation(TransMod : in SINCGARSTransModType) : boolean
<<CORBAUnion>>
SINCGARSRecvModType
RxFMVoiceMod : FMVoiceType
RxFSKMod : FSKType
<<CORBAUnion>>
SINCGARSTransModType
TxFMVoiceMod : FMVoiceType
TxFSKMod : FSKType
<<CORBAStruct>>
FMVoiceType
VoiceDeviation : VoiceDeviationType
ToneDeviation : ToneDeviationType
<<CORBAStruct>>
FSKType
MinDevHz : unsigned short
MaxDevHz : unsigned short
MinDevHz should be 9500 and
MaxDevHz should be 14000
<<CORBAStruct>>
ToneDeviationType
MinDevHz : unsigned short
MaxDevHz : unsigned short
<<CORBAStruct>>
VoiceDeviationType
MinDevkHz : unsigned short
MaxDevkHz : unsigned short
MinDevHz should be 6500 and
MaxDevHz should be 7500
MinDevkHz should be 16 and
MaxDevkHz should be 22

Figure 3-3. SINCGARS Modulation Setup

3.1.4 RadioMode.

RadioMode (Figure 3-4) as it applies to the Physical Layer, sets the Physical Layer to off, standby, operate, or test mode.

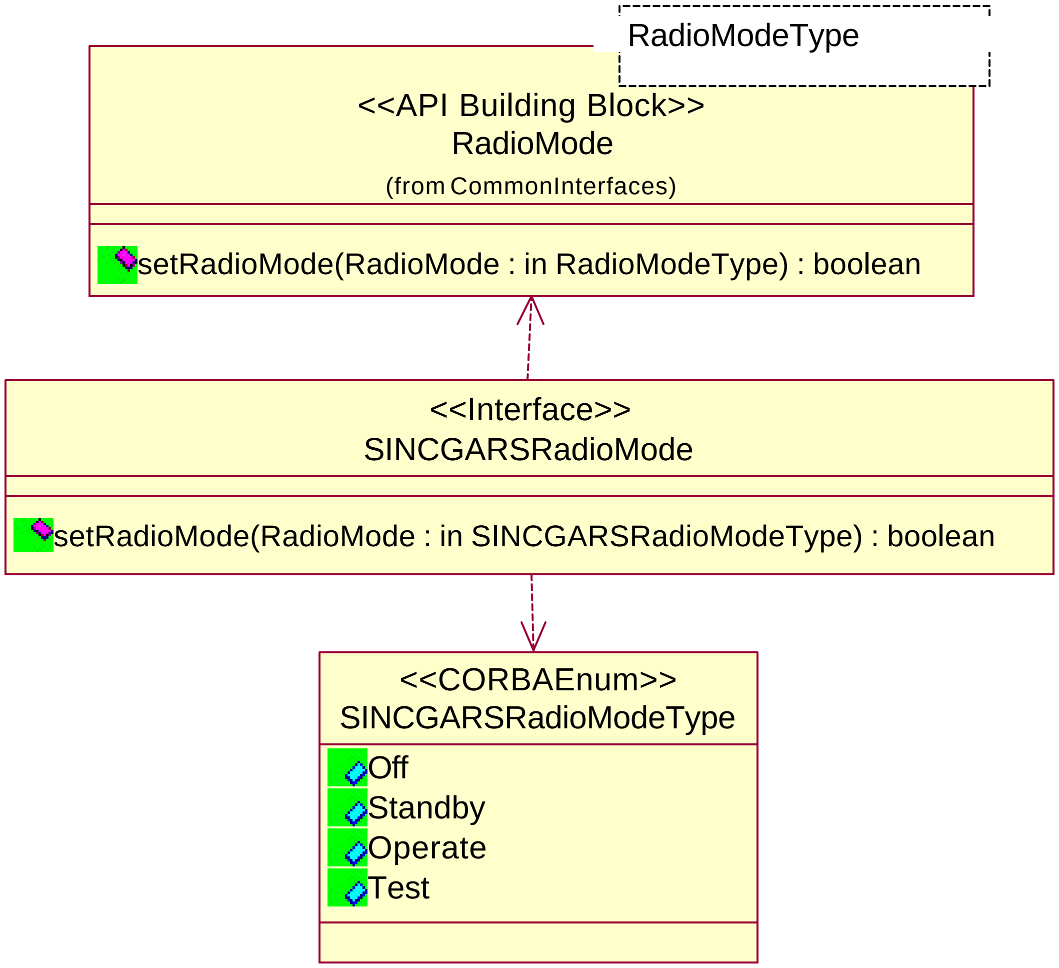

Figure 3-4. SINCGARS Radio Modes

These modes are defined as follows:

Off:

means that the waveform is no longer active on the transceiver. Power may be turned off to the Physical Layer, if possible.

Standby:

means that all TRANSEC, Crypto, and other variables are maintained, but transmit or receive is not allowed. Physical Layer put into a low power.

Operate:

is normal operation, transmit and receive operations allowed.

Test:

is a special mode of the radio to verify correct operation. Normal transmit and receive are disabled, but test transmit and testr receive can occur.

3.1.4.1 setRadioMode.

setRadioMode sets the Physical layer to either standby, operate, test, or off according the state diagram shown in Figure 3-5.

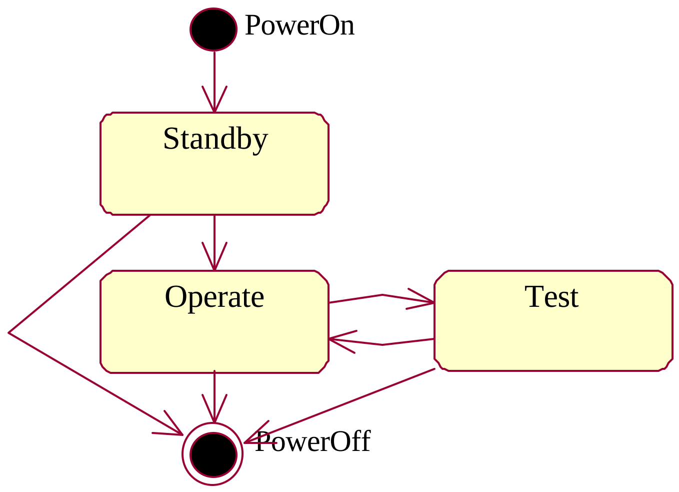

Figure 3-5. SINCGARS Radio Mode State Diagram

3.1.5 ReceiveTermination.

ReceiveTermination (Figure 3-6) inherits and instantiates ReceiveTermination directly. This service is used to terminate ongoing receptions or to disable all receptions.

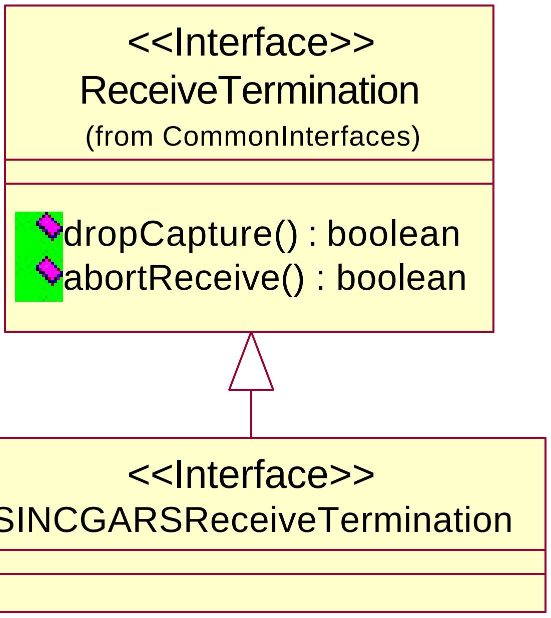

Figure 3-6. SINCGARS ReceiveTermination

3.1.5.1 dropCapture.

dropCapture terminates an ongoing reception and starts a transmit operation.

3.1.5.2 abortReceive.

abortReceive turns off the receive function.

3.1.6 TransmitInhibit.

This service group (Figure 3-7) allows the JTRS platform to enforce radio silence on specified waveform transmitters.

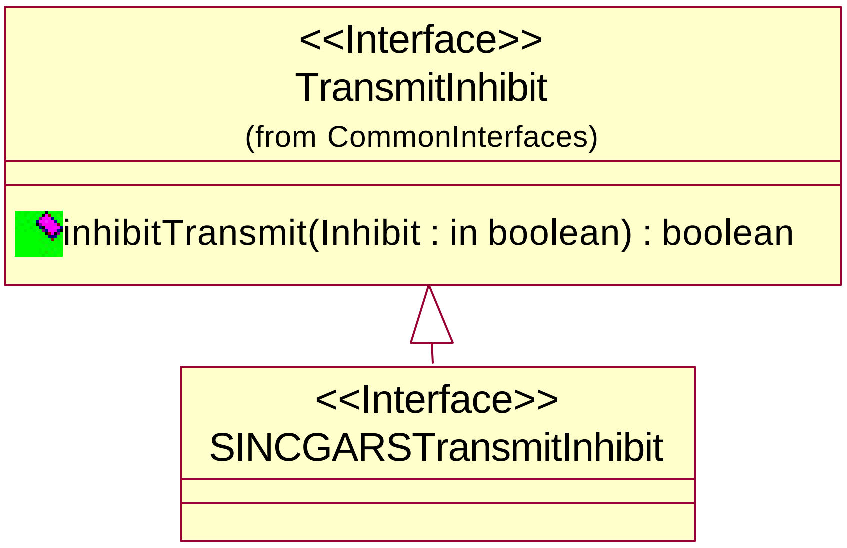

Figure 3-7. Transmit Inhibit Interface

3.1.6.1 inhibitTransmit.

inhibitTransmit is used to invoke radio silence on this SINCGARS transmitter in the JTRS platform. To invoke radio silence, the JTRS platform must invoke transmit inhibit on all active waveforms hosted on the platform.

3.1.7 PhysicalManagement.

PhysicalManagement (Figure 3-8) provides the user with the maximum and minimum transmission units that may be sent by the Physical Layer in one over-the-air transmission.

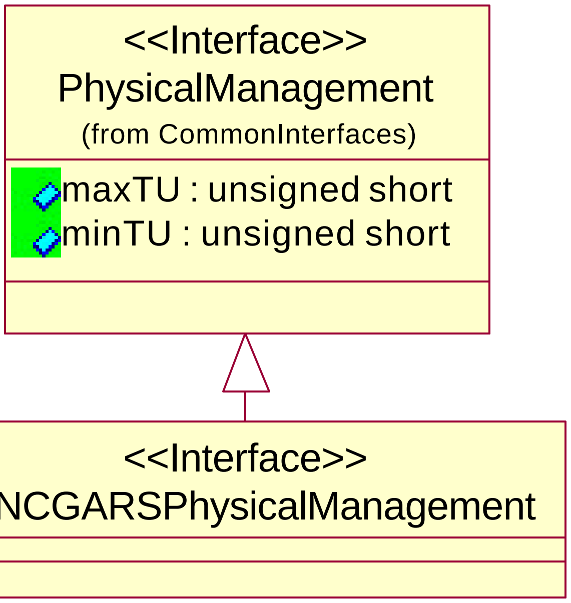

Figure 3-8. SINCGARS Physical Management

3.1.7.1 PhysicalManagement.

PhysicalManagement receives the maximum and minimum Transmission Unit(TU) sizes when the waveform is instantiated. These parameter values can not be changed.

The Transmission Unit is the over-the-air information size in octets. For packet, it is equivalent to the maximum packet size, which SINCGARS can transmit. For continuous voice or continuous data modes, this parameter is ignored since these modes can operate the radio continuously.

3.1.7.1.1 Get Minimum Number of Transmission Units Service.

Get Minimum Number of Transmission Units Service returns the minimum number of Transmission Units that can be sent in one over-the-air transmission by the Physical Layer. This is a read-only method. The SINCGARS Transmission Unit is an octet. The minimum number of Transmission Units for SINCGARS is 1800 octets.

3.1.7.1.2 Get Maximum Number of Transmission Units Service.

Get Maximum Number of Transmission Units Service returns the maximum number of Transmission Units that will be sent in one over-the-air transmission by the Physical Layer. This is a read-only method. The SINCGARS Transmission Unit is an octet. The maximum number of Transmission Units for SINCGARS is 45 octets in Single Channel and 180 octets in Frequency Hop.

3.2 REAL-TIME SERVICES.

Real-time control and data will be pushed upstream and downstream using control and data packets. The data packets include a control header for transferring real time control information with the data. These services are obtained from the Packet and Packet Signals Building Blocks. Refer to the SCA Service Definition Description for the Packet and Packet Signals Building Blocks.

In the current version of this document, exceptions are not fully specified. This should not be interpreted to mean that the methods cannot raise exceptions. Exceptions will be defined in later revisions to conform with SCA requirements and good SW Engineering practice.

3.2.1 Transmit Packet Service Group.

The SINCGARSPhysicalDownStreamProviderQueue inherits many parameters from the Packet and Packet Signals Building Blocks. The downstream Packet controls the operating frequency (receive or transmit) and provides the over-the-air data for transmit. It also provides the exact time for changing the operating frequency. Refer to Figure 3-9 for the discussion.

3.2.1.1 pushPacket Service (downstream).

The pushPacket operation is used to send real time data and frequency information down from the MAC Layer to the Physical Layer. This method identifies which priority queue the data is destined for and identifies the time for transmitting the first over-the-air symbol when required. Implicitly, this defines when the current frequency is to be updated to a new frequency and specifies the new frequency.

3.2.1.2 maxPayloadSize.

maxPayloadSize defines the maximum size of Payload in octets. This attribute allows the Service Provider to bound its internal memory usage. The Maximum Payload Size supported by the SINCGARSPhysicalDownstreamServiceProvider is 180 octets. This is a read-only parameter.

3.2.1.3 minPayloadSize.

minPayloadSize defines the minimum size of the downstream Payload in octets. The Minimum Payload Size supported by the SINCGARSPhysicalDownstreamServiceProvider is 45 octets. This is a read-only parameter.

3.2.1.4 priority.

priority defines which priority Physical queue the packet is destined for. **(The determination to use this parameter will be made later in the design process.)**

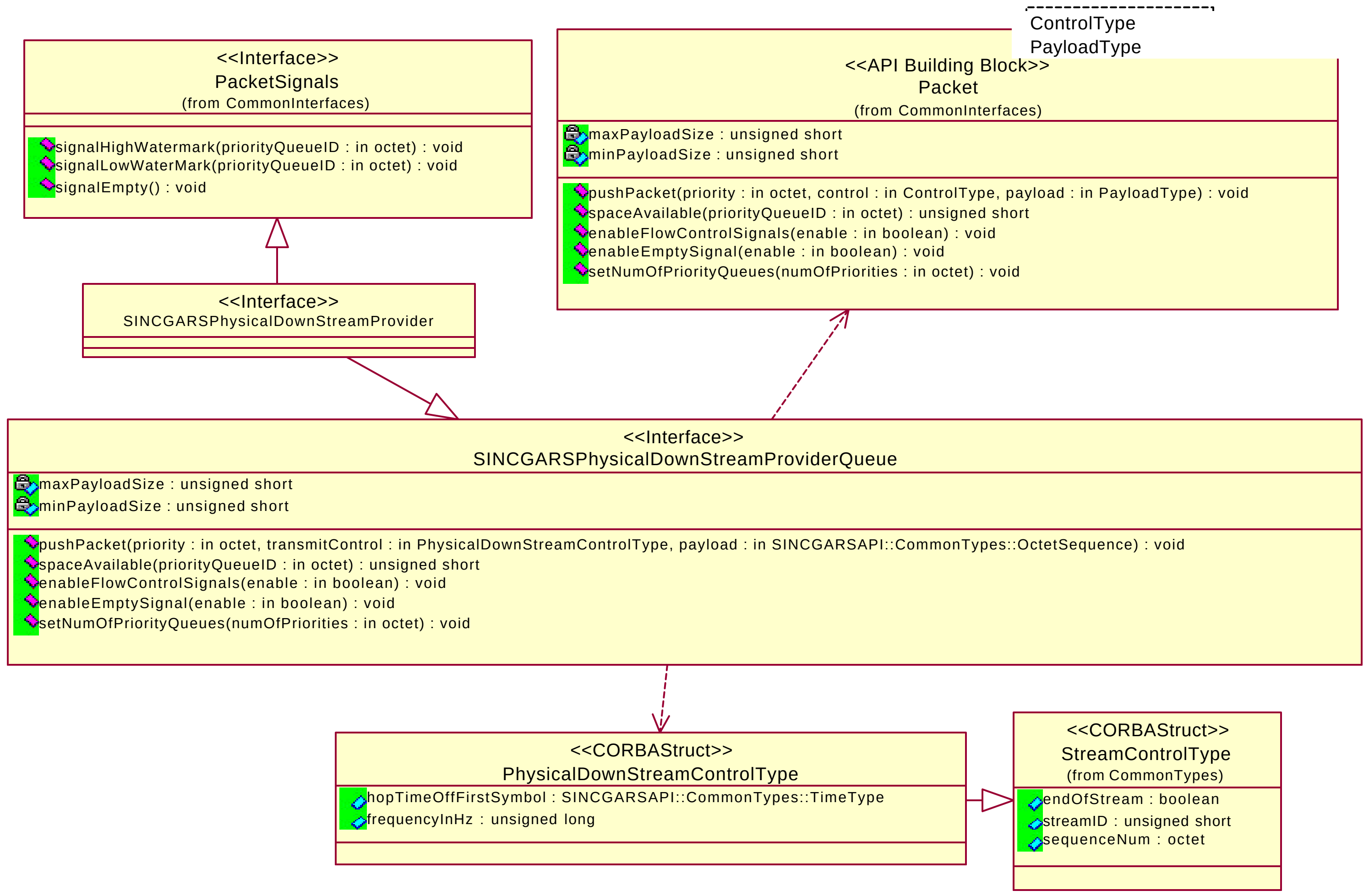
ControlType
PayloadType
<<Interface>>
PacketSignals
(from CommonInterfaces)
signalHighWatermark(priorityQueueID : in octet) : void
signalLowWaterMark(priorityQueueID : in octet) : void
signalEmpty() : void
<<API Building Block>>
Packet
(from CommonInterfaces)
maxPayloadSize : unsigned short
minPayloadSize : unsigned short
pushPacket(priority : in octet, control : in ControlType, payload : in PayloadType) : void
spaceAvailable(priorityQueueID : in octet) : unsigned short
enableFlowControlSignals(enable : in boolean) : void
enableEmptySignal(enable : in boolean) : void
setNumOfPriorityQueues(numOfPriorities : in octet) : void
<<Interface>>
SINCGARSPhysicalDownStreamProvider
<<Interface>>
SINCGARSPhysicalDownStreamProviderQueue
maxPayloadSize : unsigned short
minPayloadSize : unsigned short
pushPacket(priority : in octet, transmitControl : in PhysicalDownStreamControlType, payload : in SINCGARSAPI::CommonTypes::OctetSequence) : void
spaceAvailable(priorityQueueID : in octet) : unsigned short
enableFlowControlSignals(enable : in boolean) : void
enableEmptySignal(enable : in boolean) : void
setNumOfPriorityQueues(numOfPriorities : in octet) : void
<<CORBAStruct>>
PhysicalDownStreamControlType
hopTimeOffFirstSymbol : SINCGARSAPI::CommonTypes::TimeType
frequencyInHz : unsigned long
<<CORBAStruct>>
StreamControlType
(from CommonTypes)
endOfStream : boolean
streamID : unsigned short
sequenceNum : octet

Figure 3-9. Physical Layer DownStream Packet

3.2.1.5 hopTimeOfFirstSymbol.

hopTimeOfFirstSymbol is the time stamp in the future when transmit should begin in seconds to the nanosecond resolution. It implicitly defines when the operating frequency is to be updated to the new operating frequency.

3.2.1.6 frequencyInHz.

frequencyInHz is the frequency to tune for the duration of the hop or until updated. If set to zero, the frequency is unchanged.

3.2.1.7 streamID.

Each stream has a unique streamID (modulo the resolution of the streamID variable). **(The determination to use this parameter will be made later in the design process.)**

3.2.1.8 Sequence Number.

"sequenceNum" identifies the order of the pushPackets within a stream. The first pushPacket always has sequence number zero. **(The determination to use this parameter will be made later in the design process.)**

3.2.1.9 endOfStream.

endOfStream identifies the last pushPacket in a stream. Note that for a short stream, a single pushPacket contains both the streamID and the end of stream. **(The determination to use this parameter will be made later in the design process.)**

3.2.1.10 payload.

payload inherits octetSequence from SINCGARSAPI::CommonTypes for transferring data to the Physical Layer.

3.2.2 Downstream Flow Control Service Group.

3.2.2.1 spaceAvailable.

spaceAvailable is inherited from the Packet BB. It requests the space currently available in the specified priority queue in octets.

3.2.2.1.1 priorityQueueID.

priorityQueueID identifies the queue where space available is being interrogated.

3.2.2.2 enableFlowControlSignals.

enableFlowControlSignals enable signalHighWaterMark and signalLowWaterMark signals to be sent back to the Service User when the data in any downstream queue exceeds the High Water Mark or goes below the Low Water Mark.

3.2.2.3 enableEmptySignal.

enableEmptySignal enables the empty signal to be sent back to the Service User when all of the down-stream queues are empty.

3.2.2.4 setNumOfPriorityQueues.

setNumOfPriorityQueues sets the number of requested queues for the downstream packet. Zero is the lowest priority. **(Note the use of this method for SINCGARS *will be made later in the design process*.)**

3.2.3 Receive Packet Service Group.

The SINCGARSPhysicalUpStreamUser inherits many parameters from the Packet and Packet Signals Building Blocks. The upstream Packet contains (Figure 3-10) data received over-the-air along with the time of the first symbol and an estimate of signal strength. ***(The use of Noise Signal Strength in SINCGARS will be made later in the design process.)*** If a transmit operation is in progress, the Transmit Signal Strength is sent upstream to indicate the amount of output power.

3.2.3.1 pushPacket Service (upstream).

The pushPacket operation is used to send real-time data and control from the Physical Layer upstream to the MAC Layer. This method identifies which priority queue the data is destined for and identifies the time of receiving the first over-the-air symbol in the current pushPacket.

3.2.3.2 maxPayloadSize.

maxPayloadSize defines the maximum size of Payload in octets. This item allows the Service User to bound its internal memory usage. The Maximum Payload Size to be supported by the SINCGARSPhysicalUpStreamUser user is 45 octets. This is a read-only parameter.

3.2.3.3 minPayloadSize.

minPayloadSize defines the minimum size of Payload in octets. The Minimum Payload Size to be supported by the SINCGARSPhysicalUpStreamUser is 15 octets. This is a read-only parameter.

3.2.3.4 priority.

priority defines which priority queue the upstream packet is destined. **(The determination to use this parameter will be made later in the design process.)**

3.2.3.5 hopTimeOfFirstSymbol.

hopTimeOfFirstSymbol is the time stamp when the first symbol in the pushPacket was received.

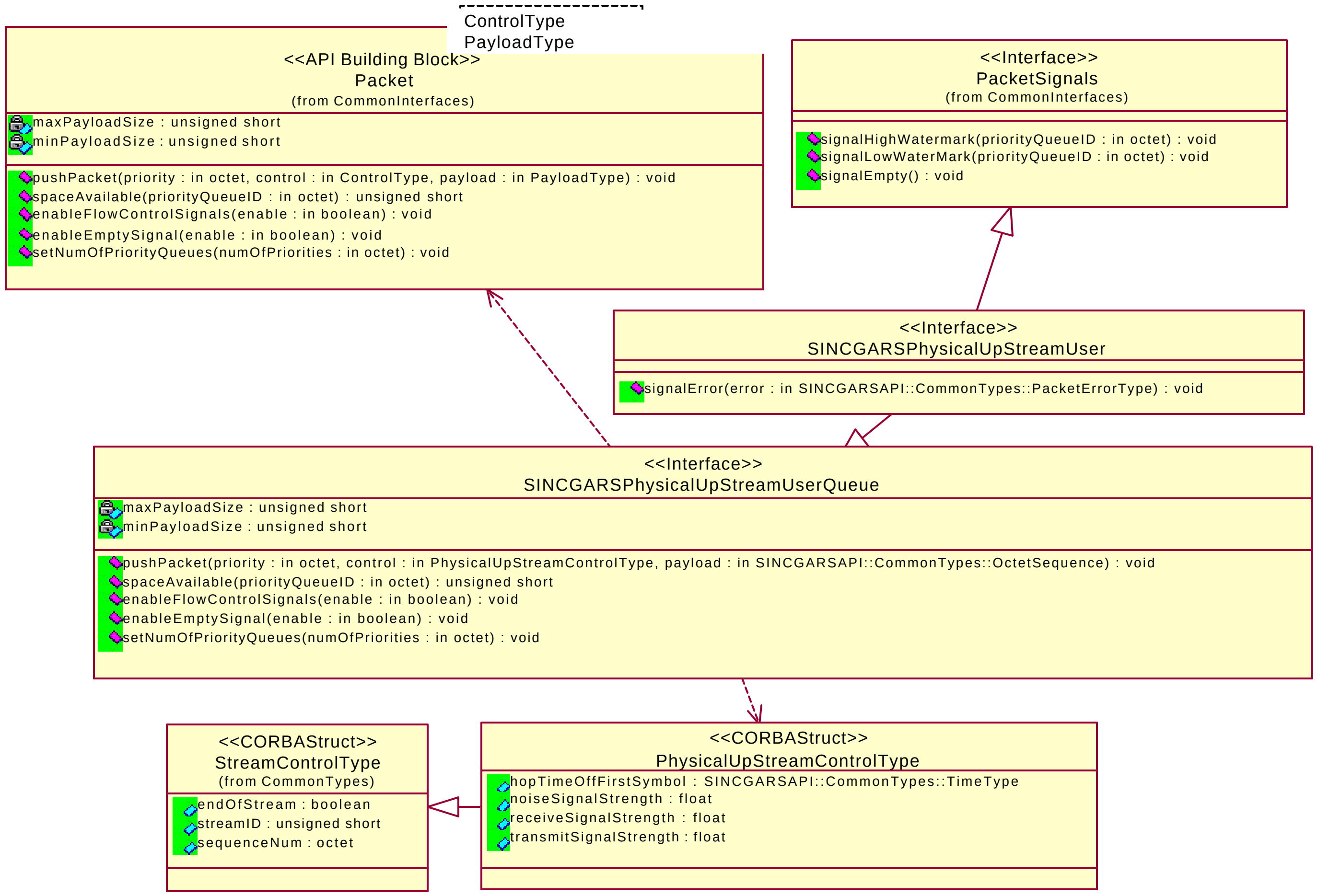
ControlType
PayloadType
<<API Building Block>>
Packet
(from CommonInterfaces)
maxPayloadSize : unsigned short
minPayloadSize : unsigned short
pushPacket(priority : in octet, control : in ControlType, payload : in PayloadType) : void
spaceAvailable(priorityQueueID : in octet) : unsigned short
enableFlowControlSignals(enable : in boolean) : void
enableEmptySignal(enable : in boolean) : void
setNumOfPriorityQueues(numOfPriorities : in octet) : void
<<Interface>>
PacketSignals
(from CommonInterfaces)
signalHighWatermark(priorityQueueID : in octet) : void
signalLowWaterMark(priorityQueueID : in octet) : void
signalEmpty() : void
<<Interface>>
SINCGARSPhysicalUpStreamUser
signalError(error : in SINCGARSAPI::CommonTypes::PacketErrorType) : void
<<Interface>>
SINCGARSPhysicalUpStreamUserQueue
maxPayloadSize : unsigned short
minPayloadSize : unsigned short
pushPacket(priority : in octet, control : in PhysicalUpStreamControlType, payload : in SINCGARSAPI::CommonTypes::OctetSequence) : void
spaceAvailable(priorityQueueID : in octet) : unsigned short
enableFlowControlSignals(enable : in boolean) : void
enableEmptySignal(enable : in boolean) : void
setNumOfPriorityQueues(numOfPriorities : in octet) : void
<<CORBAStruct>>
StreamControlType
(from CommonTypes)
endOfStream : boolean
streamID : unsigned short
sequenceNum : octet
<<CORBAStruct>>
PhysicalUpStreamControlType
hopTimeOffFirstSymbol : SINCGARSAPI::CommonTypes::TimeType
noiseSignalStrength : float
receiveSignalStrength : float
transmitSignalStrength : float

Figure 3-10. Physical Layer UpStream Packet

3.2.3.6 noiseSignalStrength.

(The determination to use this parameter will be made later in the design process.)

3.2.3.7 receiveSignalStrength.

receiveSignalStrength is the average signal strength of ***TBD*** octets of data. Zero is the weakest signal and ***TBD*** value will represent the strongest signal strength.

3.2.3.8 transmitSignalStrength.

transmitSignalStrength is an estimate of the transmit output power. Zero is the least output power and ***TBD*** represents the maximum output power.

3.2.3.9 streamID.

Each stream has a unique streamID (modulo the resolution of the streamID variable). **(The determination to use this parameter will be made later in the design process.)**

3.2.3.10 sequenceNum.

sequenceNum identifies the order of the pushPackets within a stream. The first pushPacket always has sequence number zero. **(The determination to use this parameter will be made later in the design process.)**

3.2.3.11 endOfStream.

endOfStream identifies the last pushPacket in a stream. Note that for a short stream, a single pushPacket contains both the streamID and the end of stream. **(The determination to use this parameter will be made later in the design process.)**

3.2.3.12 payload.

payload inherits octetSequence from SINCGARSAPI::CommonTypes for transferring data to the upstream Service User.

3.2.4 Upstream Flow Control Service Group.

This service group interrogates the space available in a specified queue and controls flow control signals.

3.2.4.1 spaceAvailable.

spaceAvailable is inherited from the Packet BB. It requests the space currently available in the specified priority queue in octets.

3.2.4.2 priorityQueueID.

priorityQueueID identifies the queue where space available is being interrogated.

3.2.4.3 Enable Queue Flow Control Signals.

enableFlowControlSignals enable signalHighWaterMark and signalLowWaterMark signals to be sent back to the Physical Layer service provider when the data in any up-stream queue exceeds the High Water Mark or goes below the Low Water Mark.

3.2.4.4 enableEmptySignal.

enableEmptySignal enables the empty signal to be sent back to the Physical Layer service provider when all of the upstream queues are empty.

3.2.4.5 setNumOfPriorityQueues.

setNumOfPriorityQueues sets the number of requested queues for the up-stream packet. Zero is the lowest priority.

3.2.5 PacketSignals.

The PacketSignals class is inherited from Packet Signals BB. The methods are signalHighWaterMark for the specified priority queue, signalLowWaterMark for the specified priority queue, or signal all priority queues are empty.

4 SERVICE PRIMITIVES.

This section describes the service primitives for the Physical Layer API.

4.1 NON-REAL TIME SERVICE PRIMITIVES.

Non-Real Time primitives are methods that are not time-critical for proper operation of the SINCGARS waveform.

4.1.1 Antenna Control.

4.1.1.1 setRxAntenna.

This primitive connects the receiver to one of the available antenna connectors.

4.1.1.1.1 Synopsis.

```
boolean setRxAntenna (in SINCGARSAntennaType Antenna);
```

4.1.1.1.2 Parameters

Antenna

lists all of the available SINCGARS-compatible antennas connected to the JTRS platform. The following assumes that antennas on connectors 1, 2, 3, and 4 are SINCGARS compatible and that a transmit/receive switch is provided if setRxAntenna and setTxAntenna select the same connector.

```
enum SINCGARSAntennaType{
        AntennaConnector1;
        AntennaConnector2;
        AntennaConnector3;
        AntennaConnector4;
}
```

AntennaConnector

identifies a SINCGARS-compatible antenna connector on the JTRS platform.

4.1.1.1.3 State.

This command is valid in all non-transmitting states.

4.1.1.1.4 New State.

Selected antenna connector connected to receiver.

4.1.1.1.5 Response.

TRUE if the Rx Antenna is set as specified in the call; FALSE otherwise.

4.1.1.1.6 Originator.

The service user. This is typically a command from the HCI.

4.1.1.1.7 Errors/Exceptions.

TBS.

4.1.1.2 setTxAntenna

This primitive connects the transmitter to one of the available antenna connectors.

4.1.1.2.1 Synopsis.

boolean setTxAntenna (in SINCGARSAntennaType Antenna);

4.1.1.2.2 Parameters.

Antenna

An enumeration type that lists all of the available SINCGARS-compatible antennas connected to the JTRS platform. The following assumes that antennas on connectors 1, 2, 3, and 4 are SINCGARS compatible and that a transmit/receive switch is used if setRxAntenna and setTxAntenna select the same connector.

```
enum SINCGARSAntennaType{
        AntennaConnector1;
        AntennaConnector2;
        AntennaConnector3;
        AntennaConnector4;
}
```

AntennaConnector

identifies a SINCGARS-compatible antenna connector on the JTRS platform.

4.1.1.2.3 State.

This command is valid in all states except transmit.

4.1.1.2.4 New State.

Selected antenna connector connected to transmitter.

4.1.1.2.5 Response.

TRUE if the Tx Antenna is set as specified in the call, FALSE otherwise.

4.1.1.2.6 Originator.

The service user. This is typically a command from the HCI.

4.1.1.2.7 Errors/Exceptions

None.

4.1.2 Transceiver Setup.

4.1.2.1 setUpReceiverParams.

setUpReceiverParams sets up the parameters in the Physical Layer of the transceiver that are specific to receive but are not dependent on a specific modulation.

4.1.2.1.1 Synopsis.

boolean setUpReceiverParams (in SINCGARSRecvParamsType RecvParams);

4.1.2.1.2 Parameters.

RecvParams

```
        struct SINCGARSRecvParamType{
                unsigned short          BWInKHz;
                unsigned short          carrierThreshold;
                unsigned short          bitsPerSymbol;
        }
```

BWInKHz

selects the channel bandwidth to be 25 kHz for SINCGARS.

carrierThreshold

sets the carrier threshold for not processing incoming signals. Not used for SINCGARS implementations.

bitsPerSymbol

is the number of bits to represent a symbol. In SINCGARS this is one.

4.1.2.1.3 State.

This command valid in all states.

4.1.2.1.4 New State.

This command causes a state change.

4.1.2.1.5 Response.

TRUE if all Receiver Parameters are set as specified in the call; FALSE otherwise.

4.1.2.1.6 Originator.

The service user.

4.1.2.1.7 Errors/Exceptions

None.

4.1.2.2 setUpTransmitterParams.

This primitive sets up the parameters in the transceiver Physical Layer that are specific to transmit but are not dependent on a specific modulation.

4.1.2.2.1 Synopsis.

```
boolean setUpTransmitterParams (in SINCGARSTransParamsType TransParams);
```

4.1.2.2.2 Parameters.

TransParams

sets the transmit bandwidth and the transmit envelop rise and fall times.

```
struct SINCGARSTransParamType{
        unsigned short          BWInKHz;
        unsigned long           offRampTime;
        unsigned long           onRampTime;
}
```

BWInKHz

selects the transmit channel bandwidth to be 25 kHz for SINCGARS.

offRampTime

is the time from the end of last data symbol on a hop (or the end of the last symbol in a SC message) to the end of the RF envelope.

onRampTime

is the time from the beginning of the RF envelope to the start of the first data symbol on a hop (or first symbol in a SC message).

4.1.2.2.3 State.

This command is valid in all states.

4.1.2.2.4 New State.

This command does cause a state change to the transmit envelope and bandwidth.

4.1.2.2.5 Response.

TRUE if the Transmitter Parameters are set as specified in the call, FALSE otherwise.

4.1.2.2.6 Originator.

The service user.

4.1.2.2.7 Errors/Exceptions.

None.

4.1.2.3 getBIT Results.

getBIT returns the results of the last execution of "performBIT.

4.1.2.3.1 Synopsis.

void getBIT (out SINCGARSAPI::CommonTypes::OctetSequence BITResult);

4.1.2.3.2 Parameters.

BITResult

is the result of the last performBIT. An error is returned if BIT has not been executed since waveform instantiation.

4.1.2.3.3 State.

This command is valid in test mode only.

4.1.2.3.4 New State.

This command does cause a state change.

4.1.2.3.5 Response.

Returns results of last BIT test.

4.1.2.3.6 Originator.

The service user.

4.1.2.3.7 Errors/Exceptions.

None.

4.1.2.4 performBIT.

peformBIT invokes BIT at the Physical layer.

4.1.2.4.1 Synopsis.

void performBIT{in unsigned long BITIdentifier);

4.1.2.4.2 Parameters.

BITIdentifier

identifies which BIT test is to be executed now.

4.1.2.4.3 State.

This command is valid in test mode only.

4.1.2.4.4 New State.

This command causes a state change.

4.1.2.4.5 Response.

None.

4.1.2.4.6 Originator.

The service user.

4.1.2.4.7 Errors/Exceptions.

None.

4.1.3 Modulation Setup.

4.1.3.1 setUpReceiverModulation.

This primitive sets up the parameters in the Physical Layer of SINCGARS that are specific to reception of a specific modulation.

4.1.3.1.1 Synopsis.

boolean setUpReceiverModulation (in SINCGARSRecvModType RecvMod);

4.1.3.1.2 Parameters.

RecvMod

sets up the FM and FSK demodulation peak-to-peak deviation parameters.

```
union SINCGARSRecvMod{
        FMVoiceType         RxFMVoiceMod;
        FSKType             RxFSKMod;
}

        struct FMVoiceType{
                VoiceDeviationType          VoiceDeviation;
                ToneDeviationType           ToneDeviation;
        }

                struct VoiceDeviationType{
                        unsigned short          MinDevkHz;
                        unsigned short          MaxDevkHz;
                }
```

MinDevkHz

is 16 kHz peak-to-peak at maximum deviation.

MaxDevkHz

is 22 kHz peak-to-peak at maximum deviation.

```
struct ToneDeviationType{
        unsigned short          MinDevHz;
        unsigned short          MaxDevHz;
}
```

MinDevHz
is 6500 Hz peak-to-peak.

MaxDevkHz
is 7500 Hz peak-to-peak.

```
struct  FSKType{
        unsigned short          MinDevHz;
        unsigned short          MaxDevHz;
}
```

MinDevHz
is 9500 Hz peak-to-peak maximum deviation.

MaxDevHz
is 14000 Hz peak-to-peak maximum deviation.

4.1.3.1.3 State.

This command is valid in non-transmitting states.

4.1.3.1.4 New State.

This command causes a state change.

4.1.3.1.5 Response.

TRUE if the Transmitter Modulation is set as specified in the call; FALSE otherwise.

4.1.3.1.6 Originator.

The service user.

4.1.3.1.7 Errors/Exceptions.

None.

4.1.3.2 setUpTransmitModulation.

This primitive sets up the parameters in the Physical Layer of SINCGARS that are specific to transmitting.

4.1.3.2.1 Synopsis.

boolean setUpTransmitterModulation (in SINCGARSTransModType TransMod);

4.1.3.2.2 Parameters.

TransMod

sets the maximum and minimum peak-to-peak deviation for the transmitter.

```
union SINCGARSTransModType{
        FMVoiceType         TxFMVoiceMod;
        FSKType             TxFSKMod;
}

        struct FMVoiceType{
                VoiceDeviationType        VoiceDeviation;
                ToneDeviationType         ToneDeviation;
        }

                struct VoiceDeviationType{
                        unsigned short        MinDevkHz;
                        unsigned short        MaxDevkHz;
                }
```

MinDevkHz

is 16 kHz peak-to-peak at maximum deviation.

MaxDevkHz

is 22 kHz peak-to-peak at maximum deviation.

```
                struct ToneDeviationType{
                        unsigned short        MinDevHz;
                        unsigned short        MaxDevHz;
                }
```

MinDevHz

is 6500 Hz peak-to-peak.

MaxDevkHz

is 7500 Hz peak-to-peak.

```
        struct  FSKType{
                unsigned short        MinDevHz;
                unsigned short        MaxDevHz;
        }
```

MinDevHz

is 9500 Hz peak-to-peak maximum deviation.

MaxDevHz

is 14000 Hz peak-to-peak maximum deviation.

4.1.3.2.3 State.

This command is valid in all non-transmitting states.

4.1.3.2.4 New State.

This command causes a state change to the new settings.

4.1.3.2.5 Response.

boolean

TRUE if the Transmitter Modulation is set as specified in the call; FALSE otherwise.

4.1.3.2.6 Originator.

The Service User.

4.1.3.2.7 Errors/Exceptions.

None.

4.1.4 Radio Mode.

4.1.4.1 setRadioMode.

This primitive sets the mode of the radio Physical Layer to Off, Standby, Operate, or Test.

4.1.4.1.1 Synopsis.

```
boolean  setRadioMode (in SINCGARSRadioModeType RadioMode);
```

4.1.4.1.2 Parameters.

RadioMode

sets the radio to be one of four enumerated modes.

```
enum SINCGARSRadioModeType{
        Off;
        Standby;
        Operate;
        Test;
}
```

Off

means that the waveform is no longer active on the transceiver.

Standby

means that all TRANSEC, Crypto, and other variables are maintained, but transmit or receive is not allowed. Typically a low power mode.

Operate

is normal operation, transmit and receive operations allowed.

Test

is a special mode of the radio to verify correct operation. Normal transmit and receive are disabled, but test transmit and test receive can occur.

4.1.4.1.3 State.

This command operates in all states.

4.1.4.1.4 New State.

The methods cause changes in state. The new state is specified in the command.

4.1.4.1.5 Response.

boolean

TRUE if the Radio Mode is set correctly; FALSE otherwise.

4.1.4.1.6 Originator.

This primitive is initiated by the Service User, normally through the Human Computer Interface (HCI).

4.1.4.1.7 Errors/Exceptions.

None.

4.1.5 Receive Termination.

Drop capture and abort receive are two ways to terminate the receive process.

4.1.5.1 Drop Capture.

Drop Capture zeros all state variables relating to the current signal reception and demodulation and transitions into the transmit state.

4.1.5.1.1 Synopsis.

boolean dropCapture ();

4.1.5.1.2 Parameters.

None.

4.1.5.1.3 State.

This command is valid in non-transmitting states. If the transceiver is not actively receiving a message, the radio directly transitions into transmit state.

4.1.5.1.4 New State.

This command causes a state change to transmit.

4.1.5.1.5 Response.

TRUE, if current reception is terminated or if the transceiver is not actively receiving; FALSE, if an active receive is not terminated.

4.1.5.1.6 Originator.

The service user.

4.1.5.1.7 Errors/Exceptions.

None.

4.1.5.2 abortReceive.

abortReceive turns off the receive operation.

4.1.5.2.1 Synopsis.

```
boolean   abortReceive ();
```

4.1.5.2.2 Parameters.

This primitive has no parameters.

4.1.5.2.3 State.

This command is valid in all states.

4.1.5.2.4 New State.

This command causes a state change.

4.1.5.2.5 Response.

TRUE, if receiver disabled; FALSE, if receiver not disabled.

4.1.5.2.6 Originator.

The service user.

4.1.5.2.7 Errors/Exceptions.

None.

4.1.6 Transmit Inhibit.

This is the method normally used by the JTRS radio platform to invoke radio silence on this waveform.

4.1.6.1 inhibitTransmit.

inhibitTransmit inhibits transmitting in the JTRS platform using this waveform instance.

4.1.6.1.1 Synopsis.

```
boolean   inhibitTransmit (in boolean Inhibit);
```

4.1.6.1.2 Parameters.

Inhibit

stops all waveform transmissions functions when enabled. Normal transmit operations resume when disabled.

4.1.6.1.3 State.

This command is valid in all states.

4.1.6.1.4 New State.

This command causes the waveform on the JTRS platform to go into radio silence.

4.1.6.1.5 Response.

TRUE if radio silence is achieved for the waveform transmitter; FALSE otherwise.

4.1.6.1.6 Originator.

The service user.

4.1.6.1.7 Errors/Exceptions.

None.

4.1.7 Physical Management.

Physical Management has two read-only parameters (determined at set up time) which define the minimum and maximum number of Transmission Units(TUs) the Physical Layer can accept for one over-the-air transmission. The TU is measured in octets. The minimum and maximum TUs are not adjustable parameters.

4.1.7.1 getMinTU.

Read the minimum transmission unit accepted by the Physical Layer for one over-the-air transmission.

4.1.7.1.1 Synopsis.

```
unsigned short  getMinTU ( );
```

4.1.7.1.2 Parameters.

minTU

is returned by the getMinTU call. It is the minimum number of TUs allowed in one over-the-air transmission. For SINCGARS this is 45 octets.

4.1.7.1.3 State.

This command valid in all states. This is a read-only parameter set at waveform instantiation.

4.1.7.1.4 New State.

This command does not cause a state change.

4.1.7.1.5 Response.

The size of the minimum TU that may be sent to the Physical Layer for one over-the-air transmission.

4.1.7.1.6 Originator.

The service user

4.1.7.1.7 Errors/Exceptions.

None.

4.1.7.2 getMaxTU.

Read the maximum transmission unit (TU) that will be accepted by the Physical Layer.

4.1.7.2.1 Synopsis.

Unsigned short getMaxTU ( );

4.1.7.2.2 Parameters.

maxTU

> is returned by the getMaxTU call. It is the maximum number of TUs allowed in a one over-the-air transmission. The maximum size TU for SINCGARS is 180 octets. This parameter is ignored in continuous transmit modes.

4.1.7.2.3 State.

This command is valid in all states

4.1.7.2.4 New State.

This command does not cause a state change.

4.1.7.2.5 Response.

The size of the maximum TU.

4.1.7.2.6 Originator.

The service user.

4.1.7.2.7 Errors/Exceptions

None.

4.2 REAL TIME SERVICE PRIMITIVES.

The Physical Real Time Packet Service is used to pass time-critical control and data to and from the Physical Layer. This service inherits from both the Packet Building Block and the PacketSignals Building Block. The error signals will be defined during detailed design. Since control and data are transferred to and from the Physical Layer, both a downstream Service provider interface and an upstream User interface are provided. (Downstream is transferring data toward the antenna while upstream is transferring data away from the antenna.)

4.2.1 Transmit Packet Service Group.

This group is serviced by the SINCGARSPhysicalDownStreamProvider interface.

4.2.1.1 pushPacket Service (downstream).

pushPacket is the method to move real-time data and control information into the Physical Layer.

4.2.1.1.1 Synopsis.

```
void    pushPacket (
            in octet                                        priority;
            in PhysicalDownStreamControlType                transmitControl;
            in SINCGARSAPI::CommonTypes::OctetSequence      payload;
);
```

4.2.1.1.2 Parameters.

priority

determines which priority queue the packet is destined (0 is lowest priority).

(The use of this parameter for SINCGARS will be determined later in the design.)

transmitControl

is inherited from the real-time Physical Building Block as defined below. One of its functions is to concatenate several data packets into one data stream as required. It is also supplies hop time of the first symbol and the new operating frequency.

(The use of this parameter for SINCGARS will be determined later in the design.))

```
struct PhysicalDownStreamControlType {
        SINCGARSAPI::CommonTypes::TimeType
        hopTimeOffFirstSymbol;
        unsigned long frequencyInHz;
};
```

hopTimeOfFirstSymbol

defines the local platform time at which the transmit of the first symbol in the pushPacket is to begin.

frequencyInHz

is the new operating frequency in Hertz which will be invoked a predetermined number of nanoseconds before hopTimeOfFirstSymbol. Zero frequency implies no frequency change.

```
struct  StreamControlType{
        boolean             endOfStream;
        unsigned short      streamID;
        octet               sequenceNumber;
};
```

endOfStream

indicates this packet is the last packet in the stream.

streamID

is the same for all packets in the same stream. It is different for different streams (to the extent modulo unsigned short allows).

sequenceNumber

defines the location of the packet within the stream. The first packet has sequence number zero to indicate Start of Stream.

4.2.1.1.3 State.

If flow control signals are enabled, the primitive can be invoked until a high water mark is signaled.

If flow control signals are disabled, the Service User is responsible for ensuring the Service Provider has queue space available prior to invoking pushPacket.

4.2.1.1.4 New State.

If the current pushPacket operation raises the Provider's queue to its high water mark, a high water mark signal will be returned.

4.2.1.1.5 Response

Upon receipt of a High Water Mark Signal, the Service User will inhibit further invocations of pushPacket until a Low Water Mark Signal or Empty Signal is signaled by the Service Provider.

4.2.1.1.6 Originator.

For downstream pushPacket, it is the Service User.

4.2.1.1.7 Errors/Exceptions.

None.

4.2.1.2 maxPayloadSize

maxPayloadSize is the maximum number of octets a Service Provider will accept in the Payload (data part) of a Packet. This is a read-only attribute, which can be read anytime; but is only set during Physical Layer instantiation. The Service User is obligated to constrain Payload length to be equal to or less than this size.

4.2.1.2.1 Synopsis.

attribute unsigned short maxPayloadSize;

maxPayloadSize is an attribute whose "Get" function is auto-generated.

4.2.1.2.2 Parameters.

maxPayloadSize	is the maximum number of data octets allowed in a single SINCGARS Physical downstream pushPacket. This does not include information in the control field of the pushPacket.

4.2.1.2.3 State.

Can be read anytime. The value is set at waveform instantiation and can not be changed.

4.2.1.2.4 New State.

Establishes Service User’s maximum data payload size.

4.2.1.2.5 Response.

None.

4.2.1.2.6 Originator.

None.

4.2.1.2.7 Errors/Exceptions.

None.

4.2.1.3 minPayloadSize

minPayloadSize is the minimum number of octets, acceptable to the Service Provider, in a Packet’s Payload. This is a read-only attribute, which is set during Physical Layer instantiation. It can be read anytime; but it can not be changed. The Service User is obligated to ensure Payload length is at least this size.

4.2.1.3.1 Synopsis.

attribute unsigned short minPayloadSize;

minPayloadSize is an attribute whose "Get" function is auto-generated.

4.2.1.3.2 Parameters.

minPayloadSize

is the minimum number of data octets allowed in a single SINCGARS Physical layer downstream Push Packet. This does not include information in the control field of the same Push Packet.

4.2.1.3.3 State.

Can be read anytime. The value is set at waveform instantiation and can not be changed.

4.2.1.3.4 New State.

None.

4.2.1.3.5 Response.

None.

4.2.1.3.6 Originator.

Service User.

4.2.1.3.7 Errors/Exceptions.

None.

4.2.1.4 spaceAvailable Service.

A Service Provider can provide multiple queues, each having a different priority. This read-only attribute provides a Service User with the amount of space available, in octets, in the queue designated by priorityQueueID.

4.2.1.4.1 Synopsis.

unsigned short spaceAvailable (in octet priorityQueueID);

spaceAvailable is an attribute whose "Get" function is auto-generated.

4.2.1.4.2 Parameters.

priorityQueueID

specifies which priority queue is being interrogated.

4.2.1.4.3 State.

Valid in all states.

4.2.1.4.4 New State.

No state change.

4.2.1.4.5 Response.

The return value is the space available in the Service Provider’s queue designated by the priority queue ID.

4.2.1.4.6 Originator.

Downstream Service User.

4.2.1.4.7 Errors/Exceptions.

None.

4.2.1.5 enableFlowControlSignals Service.

enableFlowControlSignals enables the Service Provider to send high water mark and low water mark signals back to the Service User for queues of all priorities.

4.2.1.5.1 Synopsis.

void enableFlowControlSignals (in boolean enable);

4.2.1.5.2 Parameters.

enable

> TRUE: permits high water mark and low water mark signals to be sent back to the downstream Service User for all priority queues.
>
> FALSE: high water mark and low water mark signals are not provided by the Service Provider.

4.2.1.5.3 State.

Valid in all states.

4.2.1.5.4 New State.

Water mark signals either enabled or disabled.

4.2.1.5.5 Response.

None.

4.2.1.5.6 Originator.

Downstream Service User.

4.2.1.5.7 Errors/Exceptions.

None.

4.2.1.6 enableEmptySignalService.

enableEmptylSignal enables the Service Provider to send an empty signal back to the Service User when all priority queues are empty.

4.2.1.6.1 Synopsis.

void enableEmptySignal (in booleanenable);

4.2.1.6.2 Parameters.

enable

> TRUE: enables the empty signal to be sent back to the downstream Service User when all queues are empty.
>
> FALSE: prevents the empty signal from being sent back.

4.2.1.6.3 State.

Valid when all priority queues are empty and empty signal is enabled.

4.2.1.6.4 New State.

No change.

4.2.1.6.5 Response.

The Service Provider sends an empty signal to the Service User when all priority queues are empty.

4.2.1.6.6 Originator.

Downstream Service User.

4.2.1.6.7 Errors/Exceptions.

None.

4.2.1.7 setNumOfPriorityQueues Service.

setNumOfPriorityQueues specifies the number of queues to be used at the downstream Service Provider.

4.2.1.7.1 Synopsis.

void setNumOfPriorityQueues (in octet numOfPriorities);

4.2.1.7.2 Parameters.

numOfPriorities

> specifies the number of priority queues to be provided by the downstream Service Provider.

4.2.1.7.3 State.

Any State.

4.2.1.7.4 New State.

The new number of queues provided by the SINCGARSPhysicalDownStreamProvider equals the value of numOfPriorities.

4.2.1.7.5 Response.

None.

4.2.1.7.6 Originator.

Downstream Service User.

4.2.1.7.7 Errors/Exceptions.

None.

4.2.2 Receive Packet Service Group

This group is serviced by the SINCGARSUpStreamUser interface.

4.2.2.1 pushPacket Service (upstream).

pushPacket is the method to send real time data and control information out of the Physical Layer to the Service User. The Service User is typically the MAC Layer.

4.2.2.1.1 Synopsis.

void pushPacket {

```
                in octet                                        priority;
                in PhysicalUpStreamControlType                  control;
                in SINCGARSAPI::CommonTypes::OctetSequence      payload;
};
```

4.2.2.1.2 Parameters.

priority

determines which priority queue the packet is destined (0 is lowest priority).

(The use of this parameter for SINCGARS will be determined later in the design.)

control

can concatenate several pushPackets into one data stream as required. It is also supplies hop time of the first received symbol in the pushPacket, the received signal strength, and transmit output power level as required.

(The use of this parameter for SINCGARS will be determined later in the design.)

```
 struct  PhysicalUpStreamControlType{
        SINCGARSAPI::CommonTypes::TimeType    hopTimeOfFirstSymbol;
        float                                 noiseSignalStrength;
        float                                 receiveSignalStrength;
        float                                 transmitSignalStrength;
        streamControlType                     streamControl;
        }
```

hopTimeOfFirstSymbol

is the local time corresponding to the start of reception of the first symbol in the pushPacket.

noiseSignalStrength

is in dB.

receiveSignalStrength

is the received signal power. Zero represents the minimum received signal power; **TBD** represents the maximum received signal power. If no signal is present, this parameter represents the noise power on frequency.

transmitSignalStrength

is the estimated transmit output power. Zero is minimum power output and **TBD** is the maximum output power.

streamControl

```
struct  StreamControlType{
        boolean             endOfStream;
        unsigned short      streamID;
        octet               sequenceNumber;
}
```

endOfStream

indicates that this packet is the last packet in the stream.

streamID

is the same for all packets in the same stream. It is different for different streams (to the extent modulo unsigned short allows).

sequenceNumber

defines the location of the packet within the stream. The first packet has sequence number zero to indicate Start of Stream.

4.2.2.1.3 State.

If flow control signals are enabled, the primitive can be invoked until a high water mark is signaled.

If flow control signals are disabled, the Service User is responsible for ensuring the Service Provider has queue space available prior to invoking pushPacket.

4.2.2.1.4 New State.

If the current pushPacket operation raises the Provider's queue to its high water mark, a high water mark signal will be returned.

4.2.2.1.5 Response.

Upon receipt of a High Water Mark Signal, the Service User will inhibit further invocations of pushPacket until a Low Water Mark Signal or Empty Signal is signaled by the Service Provider.

4.2.2.1.6 Originator.

UpStream pushPacket, it is the Service provider.

4.2.2.1.7 Errors/Exceptions.

None.

4.2.2.2 maxPayloadSize.

maxPayloadSize is the maximum number of octets a Service Provider will accept in the Payload (data part) of a Packet. This is a read-only attribute, which can be read anytime; but is only set during Physical Layer instantiation. The Service User is obligated to constrain Payload length to be equal to or less than this size.

4.2.2.2.1 Synopsis.

attribute unsigned short maxPayloadSize;

maxPayloadSize is an attribute whose "Get" function is auto-generated.

4.2.2.2.2 Parameters.

maxPayloadSize

> is the maximum number of data octets allowed in a single SINCGARS Physical upsteam pushPacket. This does not include information in the control field of the pushPacket.

4.2.2.2.3 State.

Can be read anytime. The value is set at waveform instantiation and can not be changed.

4.2.2.2.4 New State.

Establishes Service User's maximum data payload size.

4.2.2.2.5 Response.

None.

4.2.2.2.6 Originator.

None.

4.2.2.2.7 Errors/Exceptions.

None.

4.2.2.3 minPayloadSize.

minPayloadSize is the minimum number of octets, acceptable to the Service Provider, in a Packet's Payload. This is a read-only attribute, which is set during Physical Layer instantiation. It can be read anytime; but it can not be changed. The Service User is obligated to ensure Payload length is at least this size.

4.2.2.3.1 Synopsis.

attribute unsigned short minPayloadSize;

minPayloadSize is an attribute whose "Get" function is auto-generated.

4.2.2.3.2 Parameters.

minPayloadSize

> is the minimum number of data octets allowed in a single SINCGARS Physical layer upstream pushPacket. This does not include information in the control field of the same pushPacket.

4.2.2.3.3 State.

Can be read anytime. The value is set at waveform instantiation and can not be changed.

4.2.2.3.4 New State.

Establishes Service User's minimum data payload size.

4.2.2.3.5 Response.

None.

4.2.2.3.6 Originator.

None.

4.2.2.3.7 Errors/Exceptions.

None.

4.2.2.4 spaceAvailableService.

A Service Provider can provide multiple queues, each having a different priority. This read-only attribute provides a Service User with the amount of space available, in octets, in the queue designated by priorityQueueID.

4.2.2.4.1 Synopsis.

unsigned short spaceAvailable (in octet priorityQueueID);

spaceAvailable is an attribute whose "Get" function is auto-generated

4.2.2.4.2 Parameters

priorityQueueID

specifies which priority queue is being interrogated.

4.2.2.4.3 State.

Valid in all states.

4.2.2.4.4 New State.

No state change.

4.2.2.4.5 Response.

The return value is the space available in the Service Provider's queue designated by the priority queue ID.

4.2.2.4.6 Originator.

Downstream Service User.

4.2.2.4.7 Errors/Exceptions.

None.

4.2.2.5 enableFlowControlSignals Service.

enableFlowControlSignals enables the Service Provider to send high water mark and low water mark signals back to the Service User for queues of all priorities.

4.2.2.5.1 Synopsis.

void enableFlowControlSignals (in boolean enable);

4.2.2.5.2 Parameters.

enable

> TRUE: permits high water mark and low water mark signals to be sent back to the upstream Physical Service User for all priority upstream queues.
>
> FALSE: high water mark and low water mark signals are not provided by the Service Provider.

4.2.2.5.3 State.

Valid in all states.

4.2.2.5.4 New State.

Water mark signals either enabled or disabled.

4.2.2.5.5 Response.

None.

4.2.2.5.6 Originator.

Up stream, Service Provider.

4.2.2.5.7 Errors/Exceptions.

None.

4.2.2.6 enableEmptySignal Service.

enableEmptylSignal enables the Service Provider to send an empty signal back to the Service User when all priority queues are empty.

4.2.2.6.1 Synopsis

void enableEmptySignal (in booleanenable);

4.2.2.6.2 Parameters.

enable

> TRUE: enables the empty signal to be sent back to the downstream Service User when all up-stream queues are empty.
>
> FALSE: prevents the empty signal from being sent back.

4.2.2.6.3 State.

Valid when all up-stream queues are empty and the empty signal is enabled.

4.2.2.6.4 New State.

No change

4.2.2.6.5 Response.

The Service Provider sends an empty signal to the Service User when all priority queues are empty.

4.2.2.6.6 Originator.

Downstream Service User.

4.2.2.6.7 Errors/Exceptions.

None.

4.2.2.7 setNumOfPriorityQueues Service.

setNumOfPriorityQueues specifies the number of queues to be used at the downstream Service Provider.

4.2.2.7.1 Synopsis.

void setNumOfPriorityQueues (in octet numOfPriorities);

4.2.2.7.2 Parameters.

numOfPriorities,

> specifies the number of priority queues to be provided at the upstream Service User.

4.2.2.7.3 State.

Any State.

4.2.2.7.4 New State.

The new number of queues provided by the SINCGARSPhysicalUpStreamUser equals the value of numOfPriorities.

4.2.2.7.5 Response.

None.

4.2.2.7.6 Originator.

Downstream Service User.

4.2.2.7.7 Errors/Exceptions.

None.

5 ALLOWABLE SEQUENCE OF SERVICE PRIMITIVES.

There are no defined sequences of primitives.

6 UTILIZATION OF MAC BUILDING BLOCKS.

Refer to the following table describing MAC building blocks.

Services	**Parameters**	**SINCGARS SC-PT**	**SINCGARS All Except SC-PT**
Antenna Control	setRxAntenna setTxAntenna	Yes	Yes
Transceiver Setup	setReceiverParmeters setTransmiterParameters	No	No
Modulation Setup	setUPReceiverModulation setUpTransmitterModulation	Yes	Yes
Media Setup	setUpMediaType	Yes	Yes
RadioMode	setRadioMode	Yes	Yes
Receive Termination	dropCapture dropReceive	Yes	Yes
Transmit Inhibit	inhibitTransmit	No	No
Physical Management	getMaxTU getMinTU	Yes	Yes

7 PRECEDENCE OF SERVICE PRIMITIVES.

Precedence of primitives are described in section 3, where required.

8 SERVICE USER GUIDELINES.

No guidelines for implementing a Service User that will be independent of the implementation of the Service Provider have been identified.

9 SERVICE PROVIDER-SPECIFIC INFORMATION.

No specific information for a service provider implementation has been identified.

10 IDL.

The following describes the IDL software program.

10.1 COMMON TYPES.

```
//Source file: H:/JTRS/SYSJTRS/api/rose
models/ITTBBIDL/CommonTypesModules.idl

#ifndef __COMMONTYPESMODULES_DEFINED
#define __COMMONTYPESMODULES_DEFINED

/* CmIdentification
  %X% %Q% %Z% %W% */

module SINCGARSAPI {

      module CommonTypes {

            enum ServiceErrorType {
                  ERROR_BAD_SAP,
                  ERROR_BAD_ADDRESS,
                  ERROR_NO_ACCESS,
                  ERROR_INVALID_STATE,
                  ERROR_BAD_CORRELATION,
                  ERROR_BAD_DATA,
                  ERROR_UNSUPPORTED,
                  ERROR_NOT_ENABLED,
                  ERROR_TOO_MANY,
                  ERROR_BOUND,
                  ERROR_NO_AUTO,
                  ERROR_NO_XIDAUTO,
                  ERROR_NO_TESTAUTO,
                  ERROR_NO_ADDRESS,
                  ERROR_BAD_QOS_PARAMETERS,
                  ERROR_UNDELIVERABLE
            };

            /* Identify this data stream for acknowledgement
            processing, cancelation of transmission,etc. */

            struct StreamControlType {
                  unsigned short streamID;
                  /* Indicates that the last symbol of this hop is an
                  end of stream. */
                  boolean endOfStream;
                  /* Sequence number of the hop within the stream
                  sequence.  The waveform application sets this value
                  to zero at every occurrence of a start of stream. If
                  value is set to zero it indicates beginning of
                  stream. */
                  octet sequenceNum;
            };
            struct TimeType {
                  unsigned long seconds;
                  unsigned long nanoSec;
            };

            struct BeepType {
                  short BeepLevelIndB;
                  unsigned short DurationInMs;
                  unsigned short FrequencyInHz;
            };

            typedef unsigned short IdType;

            struct DescriminatorType {
                  unsigned short DataTypeDescriminator;
                  unsigned short BeepTypeDescriminator;
                  unsigned short AlarmTypeDescriminator;
                  unsigned short SeedInfoDescriminator;
            };

            enum ErrType {
                  PktUsageErr,
                  PktErrNo
            };

            union PacketErrorType switch(ErrType) {
                  case PktUsageErr:
                       SINCGARSAPI::CommonTypes::ServiceErrorType
                       usageError;
                  case PktErrNo: unsigned long errNo;
            };

            typedef sequence<octet> OctetSequence;

      };

};

#endif
```

10.2 COMMON INTERFACES.

```
//Source file: H:/JTRS/SYSJTRS/api/rose
models/ITTBBIDL/CommonInterfacesModules.idl

#ifndef __COMMONINTERFACESMODULES_DEFINED
#define __COMMONINTERFACESMODULES_DEFINED

/* CmIdentification
  %X% %Q% %Z% %W% */

module SINCGARSAPI {

      module CommonInterfaces {

            interface ChannelErrorControl {
                  /*
                  @roseuid 39EB177F0102 */
                  void channelErrorControl (
                        in boolean ErrorControl
                        );

            };

            interface DropCapture {
                  /*
                  @roseuid 39C643F203D4 */
                  boolean dropCapture ();

            };

            interface ReceiveTermination {
                  /*
                  @roseuid 39D0073C0171 */
                  boolean dropCapture ();

                  /*
                  @roseuid 39D0ADCC0174 */
                  boolean abortReceive ();

            };

            interface PhysicalManagement {
                  attribute unsigned short minTU;
                  attribute unsigned short maxTU;
            };

            interface TransmitInhibit {
                  /*
                  @roseuid 39D0B62B0021 */
                  boolean inhibitTransmit (
                        in boolean Inhibit
                        );

            };
```

```
interface PacketSignals {
      /* This operation is a call event back to the
      PacketAPI client indicating that a queue has reach
      the high watermark.  If priority or multiple queues
      are being supported then the priorityQueueID
      indicates which queue has reached the high watermark.
      @roseuid 38F3442F01B8 */
      oneway void signalHighWatermark (
            in octet priorityQueueID
            );

      /* This operation is a call event back to the
      PacketAPI client indicating that the queue has reach
      the low watermark.   If priority or multiple queues
      are being supported then this indicates that the sum
      total of all the queues has reached the low
      watermark.
      @roseuid 38F3446F025A */
      oneway void signalLowWaterMark (
            in octet priorityQueueID
            );

      /* This operation is a call event back to the
      PacketAPI client indicating that the queue has
      emptied.   If priority or multiple queues are being
      supported then this indicates that the sum total of
      all the queues has reached zero.
      @roseuid 38FE26CF02FA */
      oneway void signalEmpty ();

};

interface AudioControl {
      attribute boolean SidetoneEnabled;
      attribute short MicrophoneGainIndB;
      attribute boolean AudioOutputEnabled;
      attribute short OutputGainIndB;

      /*
      @roseuid 39ECA7A50255 */
      void enableRTSAndCTS (
            in boolean Enable
            );

      /*
      @roseuid 39ECA7C601B2 */
      void setCTS (
            in boolean CTS
            );

};

interface AudiodeviceSignals {
      /*
      @roseuid 39E4A192016E */
      void SignalRTS (
                                    in boolean RTS
                                    );

                        };

                  };

            };

            #endif
```

10.3 NON-REAL TIME.

```
//Source file: H:/JTRS/SYSJTRS/api/rose models/ITTBBIDL/NRTPhysical.idl

#ifndef __NRTPHYSICAL_DEFINED
#define __NRTPHYSICAL_DEFINED

/* CmIdentification
  %X% %Q% %Z% %W% */

#include "CommonTypesModules.idl"
#include "CommonInterfacesModules.idl"

module SINCGARSAPI {

      module NRTPhysical {

            module AntennaControl {

                  enum SINCGARSAntennaType {
                        AntennaConnector1,
                        AntennaConnector2,
                        AntennaConnector3,
                        AntennaConnector4
                  };

                  /* <Unspecified> */

                  interface SINCGARSAntennaControl {
                        /*
                        @roseuid 39DB196E0031 */
                        boolean setRxAntenna (
                              in SINCGARSAntennaType Antenna
                              );

                        /*
                        @roseuid 39DB196E0047 */
                        boolean setTxAntenna (
                              in SINCGARSAntennaType Antenna
                              );

                  };

            };

            module TRANCEIVERSetup {

                  struct SINCGARSRecvParamsType {
                        unsigned short bitsPerSymbol;
                        unsigned short carrierThreshold;
                        unsigned short BWInkHz;
                  };

                  struct SINCGARSTransParamsType {
                        unsigned short BWInkHz;
            unsigned long offRampTime;
            unsigned long onRampTime;
      };

      interface SINCGARSTranscieverSetup {
            attribute boolean BIT;

            /*
            @roseuid 39DB1D78017A */
            boolean setUpReceiverParams (
                  in SINCGARSRecvParamsType RecvParams
                  );

            /*
            @roseuid 39DB1D780184 */
            boolean setUpTransmitterParams (
                  in SINCGARSTransParamsType TransParams
                  );

            /*
            @roseuid 39DB72750168 */
            void getBIT (
                  out
                  SINCGARSAPI::CommonTypes::OctetSequence
                  BITResult
                  );

            /*
            @roseuid 39DB72D9037F */
            void performBIT (
                  in unsigned long BITIdentifier
                  );

      };

};

module ModulationSetUp {

      struct ToneDeviationType {
            unsigned short MaxDevHz;
            unsigned short MinDevHz;
      };

      struct VoiceDeviationType {
            unsigned short MinDevkHz;
            unsigned short MaxDevkHz;
      };

      struct FSKType {
            unsigned short MaxDevHz;
            unsigned short MinDevHz;
      };

      struct FMVoiceType {
            VoiceDeviationType VoiceDeviation;
```

```
                        ToneDeviationType ToneDeviation;
                };

                enum RecvModDiscriminator {
                        RxFMVoice,
                        RxFSK
                };

                union SINCGARSRecvModType
                switch(RecvModDiscriminator) {
                        case RxFSK: FSKType RxFSKMod;
                        case RxFMVoice: FMVoiceType RxFMVoiceMod;
                };

                enum TransModDiscriminator {
                        TxFMVoice,
                        TxFSK
                };

                union SINCGARSTransModType
                switch(TransModDiscriminator) {
                        case TxFSK: FSKType TxFSKMod;
                        case TxFMVoice: FMVoiceType TxFMVoiceMod;
                };

                interface SINCGARSModulationSetup {
                        /*
                        @roseuid 39DB260101A3 */
                        boolean setUpReceiverModulation (
                                in SINCGARSRecvModType RecvMod
                                );

                        /*
                        @roseuid 39DB260101A5 */
                        boolean setUpTransmitterModulation (
                                in SINCGARSTransModType TransMod
                                );

                };

        };

        module RadioMode {

                enum SINCGARSRadioModeType {
                        Off,
                        Standby,
                        Operate,
                        Test
                };

                interface SINCGARSRadioMode {
                        /*
                        @roseuid 39DB2FB101F0 */
                        boolean setRadioMode (
                                in SINCGARSRadioModeType RadioMode
                        );

                  };

            };

            module ReceiveTermination {

            interface SINCGARSReceiveTermination :
            CommonInterfaces::ReceiveTermination {
                  };

            };

            module TransmitInhibit {

            interface SINCGARSTransmitInhibit :
            CommonInterfaces::TransmitInhibit {
                  };

            };

            module PhysicalManagement {

            interface SINCGARSPhysicalManagement :
            CommonInterfaces::PhysicalManagement {
                  };

            };

      };

};

#endif
```

10.4 REAL TIME.

```
//Source file: H:/JTRS/SYSJTRS/api/rose models/ITTBBIDL/RTPhysical.idl

#ifndef __RTPHYSICAL_DEFINED
#define __RTPHYSICAL_DEFINED

/* CmIdentification
  %X% %Q% %Z% %W% */

#include "CommonTypesModules.idl"
#include "CommonInterfacesModules.idl"

module SINCGARSAPI {

      module RTPhysical {

            module PhysicalPacketTransfer {

                  struct PhysicalDownStreamControlType {
                         SINCGARSAPI::CommonTypes::TimeType
                           hopTimeOffFirstSymbol;
                         unsigned long frequencyInHz;
                  };

                  struct PhysicalUpStreamControlType {
                  SINCGARSAPI::CommonTypes::TimeType
                  hopTimeOffFirstSymbol;
                        float noiseSignalStrength;
                        float receiveSignalStrength;
                        float transmitSignalStrength;
                  };

                  interface SINCGARSPhysicalDownStreamProviderQueue {
                  /* The maxPacketSize is a read only attribute set by
                  the Packet Server and the get operation reports back
                  the maximum number of traffic units allowed in one
                  pushPacket call. */

                        attribute unsigned short maxPayloadSize;
                        attribute unsigned short minPayloadSize;

                        /* This operation is used to push Client data
                        to the Server with a Control element and a
                        Payload element.
                        @roseuid 39DC9D5A0114 */
                        void pushPacket (
                              in octet priority,
                              in PhysicalDownStreamControlType
                                 transmitControl,
                              in SINCGARSAPI::CommonTypes::
                                 OctetSequence payload
                              );
      /* The operation returns the space available in
      the Servers queue(s) in terms of the
      implementers defined Traffic Units.
      @roseuid 39DC9D5A011E */
      unsigned short spaceAvailable (
            in octet priorityQueueID
            );

      /* This operation allows the client to turn the
      High Watermark Signal ON and OFF.
      @roseuid 39DC9D5A0120 */
      void enableFlowControlSignals (
            in boolean enable
            );

      /* This operation allows the client to turn
      theEmpty Signal ON and OFF.
      @roseuid 39DC9D5A0129 */
      void enableEmptySignal (
            in boolean enable
            );

      /*
      @roseuid 39DC9D5A0132 */
      void setNumOfPriorityQueues (
            in octet numOfPriorities
            );

};

interface SINCGARSPhysicalDownStreamProvider :
SINCGARSPhysicalDownStreamProviderQueue,
CommonInterfaces::PacketSignals {
};

interface SINCGARSPhysicalUpStreamUserQueue {
/* The maxPacketSize is a read only attribute set by
the Packet Server and the get operation reports back
the maximum number of traffic units allowed in one
pushPacket call. */

      attribute unsigned short maxPayloadSize;
      attribute unsigned short minPayloadSize;

      /* This operation is used to push Client data
      to the Server with a Control element and a
      Payload element.
      @roseuid 39E4BE9B0302 */
      void pushPacket (
            in octet priority,
            in PhysicalUpStreamControlType control,
            in SINCGARSAPI::CommonTypes::
               OctetSequence payload
            );
            /* The operation returns the space available in
            the Servers queue(s) in terms of the
            implementers defined Traffic Units.
            @roseuid 39E4BE9B030E */
            unsigned short spaceAvailable (
                  in octet priorityQueueID
                  );

            /* This operation allows the client to turn the
            High Watermark Signal ON and OFF.
            @roseuid 39E4BE9B0317 */
            void enableFlowControlSignals (
                  in boolean enable
                  );

            /* This operation allows the client to turn
            theEmpty Signal ON and OFF.
            @roseuid 39E4BE9B033F */
            void enableEmptySignal (
                  in boolean enable
                  );

            /*
            @roseuid 39E4BE9B0349 */
            void setNumOfPriorityQueues (
                  in octet numOfPriorities
                  );

      };

      interface SINCGARSPhysicalUpStreamUser :
      SINCGARSPhysicalUpStreamUserQueue,
      CommonInterfaces::PacketSignals {
            /*
            @roseuid 39E4C119018D */
            void signalError (
                  in SINCGARSAPI::CommonTypes::
                     PacketErrorType error
                  );

      };

};

module CommandReceive {

      struct SINCGARSRxCommandType {
            SINCGARSAPI::CommonTypes::TimeType
            hopTimeOffFirstSymbol;
            unsigned long frequencyInHz;
      };

      interface SINCGARSReceiveCommand {
            /*
            @roseuid 39E357C1023D */
            void receive (
                              in SINCGARSRxCommandType SINCGARScontrol
                              );

                  };

            };

      };

};

#endif
```

11 UML.

UML diagrams are provided in section 3, as appropriate.